REMARKS

Reconsideration of the application is respectfully requested in view of the foregoing amendments and following remarks.

Claims 1-30 are currently pending in this application. Applicants have added claims 26-30. Claims 1-25 were rejected in the Office action dated September 17, 2003, in view of different combinations of references. Applicants respectfully disagree.

I. Amendments

Applicants have amended various claims to make editorial revisions. The amendments are not for reasons related to patentability of any of the respective claims.

II. Cited Art

Applicants make the following observations in the interest of reaching a shared understanding of the disclosures of Fesslmeier, "C++ Builder 5 Features & Benefits," U.S. Patent No. 5,515,536 to Corbett et al., U.S. Patent No. 5,987,529 to Nakamura et al., and U.S. Patent No. 5,842,220 to De Groot et al.

A. Fesslmeier, "C++ Builder 5 Features & Benefits" ["Fesslmeier"]

Fesslmeier describes various features and benefits of the product C++ Builder 5. C++ Builder 5 is a C++ development system. The system includes authoring tools. The system also includes C++ compiler technology. [*See, e.g.*, pages 7-8 – "3 The most powerful ANSI/ISO C++ compiler."]

On page 9, Fesslmeier describes features to "Simplify CORBA Distributed Object Development." One feature is "IDL Integration," in which:

> Integrated IDL simplifies CORBA development by automating IDL compilation and implementation generation. As interface definitions change, C++ implementations are *automatically kept in sync in both Client and Server projects*. IDL Syntax highlighting affirms coding and reduces errors when writing and updating interface definitions. [Fesslmeier, page 9, emphasis added.]

Applicants make two observations about this section.

First, in this section, Fesslmeier describes automatically changing C++ implementation code for both client and server projects when an interface definition changes so as *to keep the*

C++ implementation code synchronized with the interface definition. This updating of C++ code appears to be done during development by an authoring tool, not at compile time by a compiler.

Second, Applicants understand the phrase "IDL compilation" to refer to the process of creating IDL stubs and skeletons for an object. This is consistent with the CORBA specification by Object Management Group, *The Common Object Request Broker: Architecture and Specification, revision 2.3*, June 1999, available online at *http://www.omg.org/technology/documents/vault.htm*. ["OMG"] A copy of section 2.1 of the document is concurrently provided in an Information Disclosure Statement. OMG, in figures 2-2, 2-3, 2-4, and 2-5 illustrates "IDL stubs" and "static IDL skeletons." Additionally, OMG, at page 2-5 describes an interface definition defined in IDL as "used to generate the client Stubs and the object implementation skeletons." The IDL stubs and skeletons are used for static procedure calls in the CORBA distributed object system.

On page 11, Fesslmeier describes tools for creating components directly out of COM+ servers. The created components then behave like other components in the C++Builder 5 environment.

B. U.S. Patent No. 5,515,536 to Corbett et al. ["Corbett"]

Corbett describes various aspects of "dispatching interfaces." In its Background, Corbett notes, "It would be desirable to allow programs to share objects without needing to access each of the interface definitions at compile time." [Corbett, 4:16-18.] Corbett then describes various methods and operations of "dispatching interfaces." Dispatching interfaces in Corbett use late (or dynamic) binding, which in Corbett is a "technique of binding a name of a method to the code implementing the method at run time." [Corbett, 5:59-61.]

Corbett does not address automatic generation of implementation code for "dispatching interfaces." Corbett does describe, however, user involvement in setting up client code based upon type definitions. [Corbett, 12:26-32.]

C. U.S. Patent No. 5,987,529 to Nakamura et al. ["Nakamura"]

Nakamura describes a mechanism for speeding up message sending in a particular object-oriented software system. To send a message, a method of a receiver object is called using

dynamic binding, in which a caller knows the name of a method and type of a receiver object, but determines (by method lookup) the method code to be executed after compile time. [Nakamura, 1:20-30.] Nakamura describes various ways of using "dispatch tables" to speed up the process of identifying the correct receiver/method for message sending. [*See*, *e.g.*, Nakamura, 1:30 – 4:65.]

D. U.S. Patent No. 5,842,220 to De Groot et al. ["De Groot"]

De Groot describes a "class signature interface" for exposing methods and attributes of interfaces of an object class. [De Groot, 1:10-13, 3:48-58.] Suppose a server COM object class supports multiple interfaces, each interface including methods and/or attributes. De Groot describes *dynamic (or late) binding* of the interfaces as being conventional run-time negotiation between a client and the server for one of the interfaces. [De Groot, 2:40-67, 3:18-45.] In contrast, De Groot describes *static (or early) binding* of the interface to be associating the client with the interfaces of the server object during the compile phase. [De Groot, 3:18-22.] The "class signature interface" of De Groot is generated for the express purpose of facilitating early binding of client objects to interfaces. [De Groot, 10:40-56, 11:13-28.]

Regardless of the timing of the client-interface binding in De Groot (i.e., whether early/static or late/dynamic), the associations between method names and method code for the methods of the interface are made through a virtual table ["VTBL"]. [*See* De Groot, 1:41-49, 2:40-67, 11:22-28.] In other words, early/static binding and late/dynamic binding are about the timing of *client-interface negotiation* in De Groot, not the timing of the association between the names of methods and the code for the methods, which is in both cases set in a VTBL.

III. Claims 1 and 3-6

Claim 1 is directed to generating a dispatch interface implementation. Definition information defines dispatch interface features of a dispatch interface that includes dispatch methods. Based upon the definition information and programming language code (for one or more other methods), a dispatch interface implementation for operating the other methods is generated. The implementation includes executable code for: (1) the other methods, (2) a dispatch method for mapping names (of the other methods) to dispatch identifiers for *binding at*

run time, and (3) a dispatch method for calling the other methods at run time responsive to client requests.

The Office action rejects claims 1 and 3-6 as being unpatentable over Fesslmeier in view of Corbett. Applicants respectfully disagree, as the combination proposed by the Examiner is improper.

The Examiner admits that Fesslmeier does not disclose various "dispatch interface" language in claim 1. [Office action of September 17, 2003, page 3.] Applicants agree. The Examiner argues, however, that Corbett demonstrates the "dispatch interface" features and:

> It would have been obvious to one of ordinary skill in the art at the time of the invention to implement C++ Builder's compiler system with dispatch operations as found in Corbett's teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to provide interface functionality for a common interface system of dispatches." [*Id.*]

Even if, for the sake of argument, Fesslmeier *could be* modified as suggested by the Examiner, this is not enough to make the Examiner's proposed modification obvious. [MPEP 2143.01; *see also* MPEP 2142.01 and 2145.X.C and D.] In fact, the Examiner's proposed modification *changes the principle of operation* of Fesslmeier and is thus improper. [*See In re Ratti*, 270 F.2d 810, MPEP § 2143.01.] In addition, the Examiner ignores portions of Fesslmeier that lead away from claim 1.

Fesslmeier describes automatically *synchronizing* C++ implementation code for both client and server projects when an interface definition changes. [Fesslmeier, page 9.] The Examiner's proposed modification would change this principle of operation of Fesslmeier. The Examiner proposes modifying Fesslmeier according to Corbett, which would allow client and server projects *not to be synchronized* as to a current interface definition. One stated purpose of dispatching interfaces in Corbett is "to allow programs to share objects without needing to access each of the interface definitions at compile time." [Corbett, 4:16-18.] Dispatching interfaces in Corbett use late (or dynamic) binding, which in Corbett is a "technique of binding a name of a method to the code implementing the method at run time." [Corbett, 5:59-61.] Late binding as in Corbett facilitates run time association between names and code for methods of an interface, such that clients and servers need not be synchronized to the same interface definition before they begin to interoperate. [*See* Corbett, 4:12-18, 5:4-12, 12:12-32.] Using Corbett to modify

Fesslmeier (as the Examiner has done) goes against the principles of operation of both Corbett and Fesslmeier.

The Examiner also ignores portions of Fesslmeier that lead away from making the modification suggested by the Examiner. Fesslmeier describes using IDL in the context of simplifying distributed object development. Conventionally, a compiler system (such as the one in Fesslmeier) may compile IDL to generate stubs/skeletons, which are used in procedure calls such as for a remote procedure call ["RPC"]. The point of RPC/distributed object design (as in Fesslmeier) is to make access uniform across a defined interface. In contrast, one point of dispatch interfaces is to facilitate run time negotiation concerning layout and/or makeup of methods of an interface. (A dispatch interface allows in some embodiments, for example, changing of the interface by adding or deleting methods and properties after the interface is "published," such that a client and server may interoperate across the interface even though the client and server are not "in sync" as to the current definition of the interface when they begin the interoperation. *See* Al Major, COM IDL & Interface Design, Wrox Press, pp. 103-104, 1999 (previously cited in an Information Disclosure Statement).)

For at least these reasons, claim 1 should be allowable.

Claims 3-6 (which depend from claim 1) should also be allowable, but Applicants will not belabor the merits of the separate patentability of claims 3-6.

IV. Claim 2

The Office action rejects claim 2 (which depends from claim 1) as being unpatentable over Fesslmeier in view of Corbett and U.S. Patent No. 5,946,489 to Yellin et al. ["Yellin"]. Applicants respectfully disagree. The combination of Fesslmeier with Corbett and Yellin is improper for at least the reasons that the combination of Fesslmeier with Corbett is improper (as explained above). Applicants will not belabor the merits of the separate patentability of claim 2. Claim 2 should be allowable.

V. Claims 7-11

Claim 7 is directed to a compiler system that generates a late binding interface implementation. The compiler system includes a front end module, a converter module, and a back end module. The front end module receives definition information (defining late binding

interface features of a late binding interface) and programming language code (for implementing one or more late bound methods). The converter module identifies relations between the definition information and the late bound methods. The back end module generates a late binding interface implementation based upon the relations, for operating the late bound methods.

The Office action rejects claims 7-11 as being unpatentable over Fesslmeier in view of Nakamura. Applicants respectfully disagree. For at least the following reasons, claim 7 should be allowable.

A. Fesslmeier and Nakamura, taken separately or in combination, fail to teach or suggest at least one limitation of claim 7.

Claim 7 recites at least one limitation that Fesslmeier and Nakamura, taken separately or in combination, fail to teach or suggest. For example, claim 7 recites a "compiler system" that includes a "converter module that identifies relations between the definition information and the one or more late bound methods" and "a back end module that generates a late binding interface implementation based upon the relations." The portions of Fesslmeier (pages 9 and 11) cited against the above-cited language involve updating C++ code based upon IDL using an authoring tool. Fesslmeier does not anywhere teach or suggest a compiler system that identifies relations between definition language and methods and generates an interface implementation based upon the relations.

Nakamura, in turn, does not relate to "definition information," and is even further from teaching or suggesting a compiler system that works with definition information as recited in claim 7.

Because Fesslmeier and Nakamura taken separately fail to teach or suggest the above-cited language of claim 7, the combination of Fesslmeier and Nakamura also fails to teach or suggest the above-cited language of claim 7, and claim 7 should be allowable.

B. The combination of Fesslmeier and Nakamura is improper.

The combination proposed by the Examiner to reject claim 7 is improper. The Examiner admits that Fesslmeier does not disclose various "late bound" language in claim 7. [Office action of September 17, 2003, page 7.] Applicants agree. The Examiner argues, however, that Nakamura demonstrates the "late bound" features and:

> It would have been obvious to one of ordinary skill in the art at the time of the invention to implement C++ Builder's system of programming using integrated IDL with the ability to create interfaces for late bound situations as found in Nakamura's teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to develop interfaces for as many situations as possible, including late bound. Furthermore, C++ Builder illustrated the use of COM interfaces." [*Id*, citations omitted.]

Even if, for the sake of argument, Fesslmeier *could be* modified as suggested by the Examiner, this is not enough to make the Examiner's proposed modification obvious. [MPEP 2143.01; *see also* MPEP 2142.01 and 2145.X.C and D.] In fact, the Examiner's proposed modification *changes the principle of operation* of Fesslmeier and is thus improper. [*See In re Ratti*, 270 F.2d 810, MPEP § 2143.01.] In addition, the Examiner ignores portions of Fesslmeier that lead away from claim 7.

Fesslmeier describes automatically *synchronizing* C++ implementation code for both client and server projects when an interface definition changes. [Fesslmeier, page 9.] The Examiner's proposed modification would change this principle of operation of Fesslmeier. The Examiner proposes modifying Fesslmeier according to Nakamura, which would allow client and server projects *not to be synchronized* as to a current interface definition. Dynamic binding in Nakamura facilitates late association between method names and method code, such that clients and servers need not be synchronized to the same interface definition before they begin to interoperate. Using Nakamura to modify Fesslmeier (as the Examiner has done) goes against the principles of operation of both Nakamura and Fesslmeier.

The Examiner also ignores portions of Fesslmeier that lead away from making the modification suggested by the Examiner. Fesslmeier describes using IDL in the context of simplifying distributed object development. Conventionally, a compiler system (such as the one in Fesslmeier) may compile IDL to generate stubs/skeletons, which are used in procedure calls such as for a RPC. The point of RPC/distributed object design (as in Fesslmeier) is to make access uniform across a defined interface. In contrast, one point of late binding interfaces is to facilitate run time negotiation concerning layout and/or makeup of methods of an interface.

Claims 8-11 (which depend from claim 7) should also be allowable, but Applicants will not belabor the merits of the separate patentability of claims 8-11.

VI. Claim 12

The Office action rejects claim 12 (which depends from claim 7) as being unpatentable over Fesslmeier in view of Nakamura and De Groot. Applicants respectfully disagree.

Taken separately or in combination with the other references, De Groot does not teach or suggest the above-cited language of claim 7 that Fesslmeier and Nakamura fail to teach or suggest. The combination of Fesslmeier with Nakamura and De Groot is improper for at least the reasons that the combination of Fesslmeier with Nakamura is improper (as explained above).

Applicants will not belabor the merits of the separate patentability of claim 12. Claim 12 should be allowable.

VII. Claims 13-16 and 18

The Office action rejects claims 13-16 and 18 as being unpatentable over Fesslmeier in view of Nakamura. Applicants respectfully disagree, as the combination proposed by the Examiner to reject claim 13 is improper.

The Examiner admits that Fesslmeier does not address various "late bound" language in claim 13. [Office action of September 17, 2003, page 9.] Applicants agree. The Examiner argues, however, that Nakamura demonstrates the "late bound" features and:

> It would have been obvious to one of ordinary skill in the art at the time of the invention to implement C++ Builder's system of programming using integrated IDL with the ability to create interfaces for late bound situations as found in Nakamura's teaching. This implementation would have been obvious because one of ordinary skill in the art would be motivated to develop interfaces for as many situations as possible, including late bound. Furthermore, C++ Builder illustrated the use of COM interfaces." [*Id*, citations omitted.]

Even if, for the sake of argument, Fesslmeier *could be* modified as suggested by the Examiner, this is not enough to make the Examiner's proposed modification obvious. [MPEP 2143.01; *see also* MPEP 2142.01 and 2145.X.C and D.] In fact, the Examiner's proposed modification *changes the principle of operation* of Fesslmeier and is thus improper. [*See In re Ratti*, 270 F.2d 810, MPEP § 2143.01.] In addition, the Examiner ignores portions of Fesslmeier that lead away from claim 13.

Fesslmeier describes automatically *synchronizing* C++ implementation code for both client and server projects when an interface definition changes. [Fesslmeier, page 9.] The Examiner's proposed modification would change this principle of operation of Fesslmeier. The

Examiner proposes modifying Fesslmeier according to Nakamura, which would allow client and server projects *not to be synchronized* as to a current interface definition. Dynamic binding in Nakamura facilitates late association between method names and method code, such that clients and servers need not be synchronized to the same interface definition before they begin to interoperate. Using Nakamura to modify Fesslmeier (as the Examiner has done) goes against the principles of operation of both Nakamura and Fesslmeier.

The Examiner also ignores portions of Fesslmeier that lead away from making the modification suggested by the Examiner. Fesslmeier describes using IDL in the context of simplifying distributed object development. Conventionally, a compiler system (such as the one in Fesslmeier) may compile IDL to generate stubs/skeletons, which are used in procedure calls such as for a RPC. The point of RPC/distributed object design (as in Fesslmeier) is to make access uniform across a defined interface. In contrast, one point of late binding interfaces is to facilitate run time negotiation concerning layout and/or makeup of methods of an interface.

For at least these reasons, claim 13 should be allowable.

Claims 14-16 and 18 (which depend from claim 13) should also be allowable, but Applicants will not belabor the merits of the separate patentability of claims 14-16 and 18.

VIII. Claim 17

The Office action rejects claim 17 (which depends from claim 13) as being unpatentable over Fesslmeier in view of Nakamura and Yellin. Applicants respectfully disagree. The combination of Fesslmeier with Nakamura and Yellin is improper for at least the reasons that the combination of Fesslmeier with Nakamura is improper (as explained above). Applicants will not belabor the merits of the separate patentability of claim 17. Claim 17 should be allowable.

IX. Claim 19

The Office action rejects claim 19 (which depends from claim 13) as being unpatentable over Fesslmeier in view of Nakamura and De Groot. Applicants respectfully disagree. The combination of Fesslmeier with Nakamura and De Groot is improper for at least the reasons that the combination of Fesslmeier with Nakamura is improper (as explained above). Applicants will not belabor the merits of the separate patentability of claim 19. Claim 19 should be allowable.

X. Claims 20-22

Claim 20 is directed to automatically generating an interface implementation having early binding and late binding mechanisms. Definition information defines late binding interface features of the interface. Based upon the definition information and programming language code (for one or more dual bound methods of the interface), an interface implementation is generated for alternatively operating the dual bound methods by an early binding mechanism or by a late binding mechanism. The early binding mechanism provides for direct invocation of the dual bound methods. (*See*, *e.g.*, page 3 of the application, which describes the use of "virtual function tables" in early binding of method names to method code in an interface.) The late binding mechanism provides for invocation of the dual bound methods responsive to a request through a late binding method.

The Office action rejects claims 20-22 as being unpatentable over Fesslmeier in view of De Groot. Applicants respectfully disagree. For at least the following reasons, claim 20 should be allowable.

Claim 20 recites at least one limitation that Fesslmeier and De Groot, taken separately or in combination, fail to teach or suggest. For example, claim 20 recites "based upon the programming language code and the definition information, generating an interface implementation for alternatively operating the one or more dual bound methods by an early binding mechanism or by a late binding mechanism." The Examiner appears to acknowledge that Fesslmeier does not teach or suggest alternatively operating methods by an early binding mechanism or by a late binding mechanism. [Office action of September 17, 2003, page 14.]

De Groot, in turn, describes using either early/static binding or late/dynamic binding of client objects to interfaces. First, this relates to using *either* early binding *or* late binding in a given implementation, not having both for use in the alternative in a given implementation. Second, this relates to binding of client objects to interfaces, not binding for method calls. In fact, for both early/static and late/dynamic client-interface binding, De Groot describes using VTBLs, which leads directly away from using a late binding mechanism for invocation of methods responsive to a request through a late binding method (as recited in claim 20).

Because Fesslmeier and De Groot taken separately fail to teach or suggest the above-cited language of claim 20, the combination of Fesslmeier and De Groot also fails to teach or suggest the above-cited language of claim 20, and claim 20 should be allowable.

Claims 21 and 22 (which depend from claim 20) should also be allowable, but Applicants will not belabor the merits of the separate patentability of claims 21 and 22.

XI. Claims 23-25

Claim 23 is directed to automatically generating call site code for calling a late bound method of a late binding interface. Based upon type information for one or more input arguments of the late bound method, code is generated for packing the input arguments into a generic argument data structure. Code is also generated for calling the late bound method through an invocation method of the late binding interface, wherein the calling includes passing the generic argument data structure to the invocation method.

The Office action rejects claims 23-25 as being unpatentable over Fesslmeier in view of De Groot. Applicants respectfully disagree.

Claim 23 recites at least one limitation that Fesslmeier and De Groot, taken separately or in combination, fail to teach or suggest. For example, claim 23 recites "based upon type information for one or more input arguments of the late bound method, generating code for packing the one or more input arguments into a generic argument data structure" and "generating code for calling the late bound method through an invocation method of the late binding interface, wherein the calling includes passing the generic argument data structure to the invocation method." The Examiner appears to acknowledge that Fesslmeier does not anywhere teach or suggest the above-cited language. [Office action of September 17, 2003, pages 15 and 16.]

De Groot, in turn, describes using either early/static binding or late/dynamic binding of client objects to interfaces. This relates to binding of client objects to interfaces, not binding for method calls. In fact, for both early/static and late/dynamic client-interface binding, De Groot describes using VTBLs, which leads directly away from the above cited language of claim 23.

Because Fesslmeier and De Groot taken separately fail to teach or suggest the above-cited language of claim 23, the combination of Fesslmeier and De Groot also fails to teach or suggest the above-cited language of claim 23, and claim 23 should be allowable.

Claims 24 and 25 (which depend from claim 23) should also be allowable, but Applicants will not belabor the merits of the separate patentability of claims 24 and 25.

XII. Form 1449 for June 15, 2001 IDS

Applicants request that the Examiner provide an initialed Form 1449 for the Information Disclosure Statement filed June 15, 2001.

CONCLUSION

For the reasons stated above, claims 1-30 should be allowed. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

By ______________________
Kyle B. Rinehart
Registration No. 47,027

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446